Representation Learning

Il concetto di **apprendimento di rappresentazione** (representation learning) lega insieme la maggior parte delle forme di apprendimento profondo (deep learning). Le reti feedforward e ricorrenti, auto-encoders e modelli probabilistici profondi apprendono e sfruttano le rappresentazioni. Apprendere la miglior possibile rappresentazione rimane la direzione promettente per quanto riguarda la ricerca.

Apprendimento di rappresentazione è la **capacità del modello di imparare una nuova rappresentazione dei dati, diversa da quella di input, che semplifica l’utilizzo dei dati successivo** (dipende dal task che si vuole risolvere).

I motivi per cui è importante apprendere le rappresentazioni sono molteplici.

Apprendere una buona rappresentazione permette la **condivisione** **delle proprietà statistiche dei dati** quando si affrontano task diversi.

L’idea: **si apprende dai dati una buona rappresentazione per i dati stessi e poi la si usa per task differenti**.

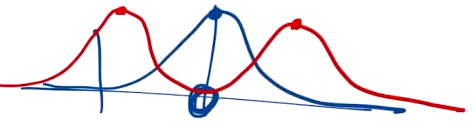
Ad esempio, nell’image processing, una rete convoluzionale profonda è costituita da una serie di livelli che apprendono una rappresentazione a cui è collegato un classificatore (in genere una rete feedforward di un paio di livelli full-connected). Questa rappresentazione viene utilizzata anche per scopi diversi dallo scopo originario per cui era stata appresa.

Le rappresentazioni condivise sono utili per **trattare molteplici modalità o domini** (quando i dati si concentrano su più modalità).

Esempio: si consideri la distribuzione normale gaussiana dei dati.

Nel caso di una singola modalità, si è in presenza di una sola gaussiana → una sola moda principale (“vetta”) su cui si concentrano le predizioni.

Spesso nei dati reali esistono situazioni in cui sono presenti più di una “vetta”. Questo è il caso di più modalità con più gaussiane.



L’apprendimento della rappresentazione dei dati, in particolare non supervisionato, permette di modellare meglio le “vette” per risolvere il task utilizzando la rappresentazione espressiva.

Le rappresentazioni condivise sono utili per **trasferire la conoscenza** **appresa** a problemi senza esempi oppure con pochi esempi ma per cui esiste una rappresentazione buona dei dati. Si parla di **Semi-supervised Learning** e **Zero-shot Learning**. Sono due termini che indicano casi più o meno estremi in cui il numero di esempi etichettati è molto basso.

**Semi-supervised Learning**: combina una piccola quantità di dati etichettati con una grande quantità di dati senza etichetta durante l'allenamento.

* tanti esempi non supervisionati (senza etichetta)
* pochi esempi supervisionati (con etichetta)

In questo contesto, se si è in grado di apprendere una buona rappresentazione partendo dai dati non supervisionati, poi si ha qualche chance per fare un buon lavoro quando si fa apprendimento sui pochi esempi supervisionati.

**Zero-shot Learning**: al momento del test uno studente osserva campioni di classi che non sono stati osservati durante l'allenamento e deve prevedere la categoria di appartenenza.

L’unico modo per affrontare questo tipo di problemi è apprendere una buona rappresentazione dei dati.

# Rappresentazioni e difficoltà

Spesso cambiare la rappresentazione può trasformare un problema molto difficile in un problema molto più semplice.

Un esempio è il **kernel trick**: cambio di rappresentazione. I dati sono mappati in un nuovo spazio in cui il problema da risolvere è risolvibile in modo lineare, la soluzione è più semplice da trovare rispetto allo spazio originario.

Dividere 2 numeri può essere semplice quando sono espressi con una notazione conveniente (es. notazione posizionale) e può essere difficile con una rappresentazione diversa (es. rappresentazione in numeri romani). Nell’ultimo caso, si convertono i numeri romani in numerazione posizionale.

Si vuole inserire un numero nella posizione corretta. Con una rappresentazione a lista ordinata, il costo delle operazioni è O(n). Con la rappresentazione ad albero rosso-nero, il costo diventa O(log n).

Il problema si semplifica in base al modo in cui si rappresentano i dati.

Trovare una buona rappresentazione è molto importante e semplifica molto spesso il problema da risolvere.

Una buona rappresentazione rende i task successivi più facili.

Addestrando le reti feedforward con un algoritmo supervisionato, possono essere pensate come oggetti che formano una qualche rappresentazione al termine della rete stessa.

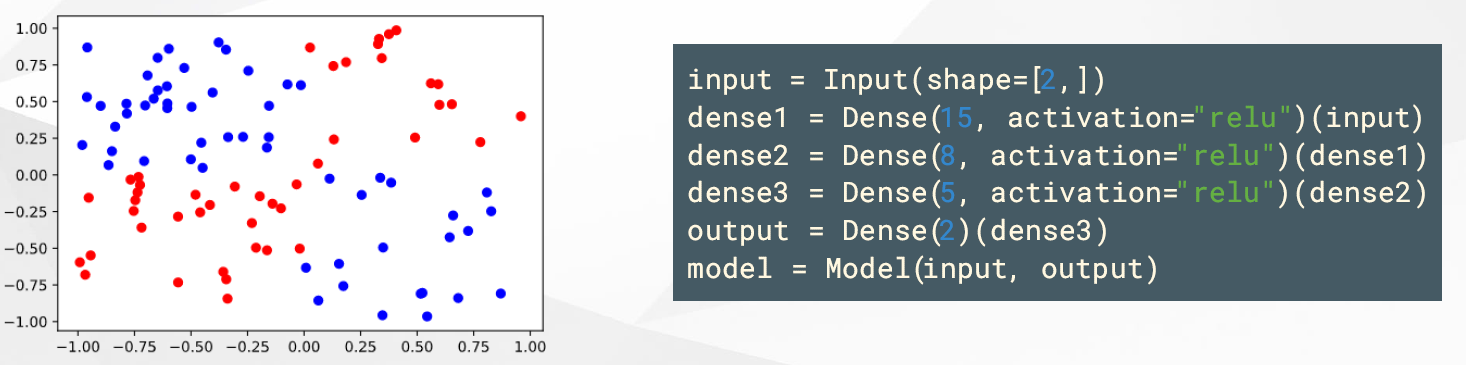
L’ultimo livello è tipicamente un classificatore semplice che può essere molto accurato perché il resto della rete lo fornisce con una buona rappresentazione.

Esempio: rete feedforward fully-connected

|  |
| --- |
| Output |
| Layer 2 → ogni neurone è rappresentato da un numero (nuova rappresentazione dell’input). Il layer 2 è fully-connected con l’output. La nuova rappresentazione permette di risolvere il task con una sola operazione. Ciò non era vero se partivo dalla rappresentazione iniziale (input). |
| Layer 1 |
| Input |

Esempio: rete convoluzionale → ogni livello di convoluzione della rete è una nuova rappresentazione. Sono reti fully-connected che possono essere pensate come classificatori costruiti sull’ultima rappresentazione (ultimo livello di convoluzione). L’ultima rappresentazione si può mettere su un nuovo dataset su disco e utilizzare una qualsiasi altra forma di classificatore (svm, decision tree…) e fare classificazione usando questa rappresentazione. Si ottiene un risultato molto vicino al risultato ottenuto utilizzando l’intera rete convoluzionale, una volta che i livelli sono addestrati correttamente.

**Esempio: XOR dataset**



Non è possibile dividere in modo lineare questo dataset.

Con Keras è stata costruita una rete feedforward: ogni livello ha unità Relu connesse.

Input → 2 neuroni

Dense 1 → 15 neuroni Dense = fully-connected

Dense 2 → 8 neuroni

Dense 3 → 5 neuroni

Output → 2 neuroni

Task: separare i punti blu dai punti rossi.

L’output ha la stessa dimensionalità dell’input: si vuole poter plottare in 2D la rappresentazione costruita.

Nel video (slide) è mostrato come cambia la rappresentazione presente nell’ultimo livello man mano che procede l’apprendimento. Inizialmente i punti sono sovrapposti. Durante l’apprendimento, la rete distorce le rappresentazioni in modo tale da distribuire i punti rossi in zone dello spazio lontane dai punti blu. Al termine sarà semplice con un modello lineare separare i punti rossi da quelli blu.

# Obiettivi multipli

Per apprendere una buona rappresentazione, bisogna tenere conto che non sempre cercare di apprendere una rappresentazione porti a buoni risultati, a meno che non si impone qualche proprietà sulla rappresentazione.

Negli auto-encoders: l’auto-encoder cerca di “copiare” l’input nell’output in modo identico, costruendo una rappresentazione intermedia dei dati e definendo su di essa dei vincoli e delle proprietà. Se non si ponessero vincoli, potrebbe succedere che tutti i neuroni apprendano la funzione identità.

Trade-off tra i seguenti obiettivi:

* preservare tutta l’informazione riguardo all’input
* forzare proprietà interessanti sulle rappresentazioni (es. indipendenza delle features)

Questi due obiettivi sono in contrapposizione l’un l’altro.

# Supervised and Unsupervised Tasks

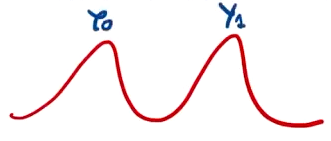
L’apprendimento semi-supervisionato (Semi supervised Learning) è spesso utile per risolvere problemi in cui il numero di esempi etichettati è piccolo.

Combina una piccola quantità di dati etichettati con una grande quantità di dati senza etichetta durante l'allenamento.

* tanti esempi non supervisionati (senza etichetta)
* pochi esempi supervisionati (con etichetta)

Un modo per procedere è apprendere buone rappresentazioni per i dati non supervisionati e poi usare queste rappresentazioni per risolvere il task di apprendimento supervisionato.

Può succedere che la rappresentazione dei dati abbia un andamento che correla con le etichette.



In Y0 e Y1 sono concentrati rispettivamente tutti gli esempi etichettati con y0 e y1.   
Se questo è vero, sono in ottima condizione per classificare bene gli esempi.

**Si impara la struttura**, ovvero come è fatta la **distribuzione che genera i dati**. Quando la distribuzione ha una struttura interna che correla con le etichette che abbiamo nella parte supervisionata, questo semplifica poi il problema supervisionato.

Greedy Layer-Wise Unsupervised Pre-Training

Una tecnica molto importante nell’ambito dell’apprendimento delle rappresentazioni è il **Greedy Layer-Wise Unsupervised Pre-Training**.

Il Greedy Layer-Wise Unsupervised Pre-Training è una tecnica che affonda le sue radici una ventina d’anni fa, è stata una tecnica strumentale per far rinascere l’interesse sulle reti neurali profonde. Per un lungo periodo di tempo si pensava che addestrare reti neurali profonde fosse troppo difficile, richiedesse troppi dati, troppe risorse computazionali e i problemi riguardanti il calcolo del gradiente che tendeva a sparire o esplodere (vanishing/exploding gradient). Si pensava che era troppo difficile per le applicazioni pratiche. Per tanto tempo la ricerca sulle reti neurali profonde è stata abbandonata.

Questa tecnica di apprendimento di rappresentazione è stata la prima che ha messo in evidenza come fosse possibile addestrare delle reti anche molto profonde con i vincoli del mondo reale, quindi con un numero di esempi non esagerato ed un potere computazionale limitato.

L’apprendimento delle rappresentazioni permette di trasferire ciò che ho appreso in un task in un altro task differente. Questa idea è molto chiara in questa tecnica.

In questa tecnica si apprendono dei blocchi in modo non supervisionato (sostanzialmente auto-encoders) che poi sono messi in cascata per risolvere un problema supervisionato.

Questo regime di apprendimento è un esempio chiaro di come apprendere una buona rappresentazione per un task (apprendimento non supervisionato, provando a catturare la forma della distribuzione) può a volte essere utile per un altro task (apprendimento supervisionato con lo stesso dominio di input).

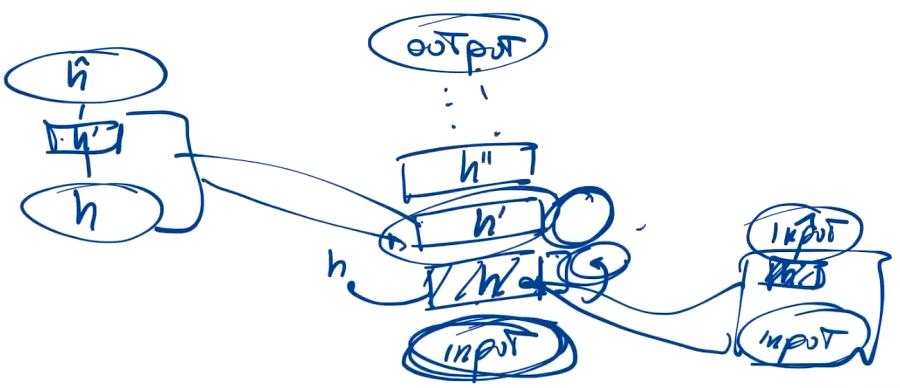
Per funzionare, questa tecnica fa leva su qualche procedura **shallow**. L’idea è imparare una architettura profonda e, per farlo, bisogna essere in grado di imparare un’architettura “piatta” come ad esempio:

* Restricted Boltzman Machines: modelli piatti completamente connessi che hanno collegamenti molto forti con la teoria dei modelli probabilistici. Sono uno dei modelli probabilistici che usano tecniche di backprop.
* Single layer auto-encoders:
  + output
  + h
  + input
* altri modelli che apprendono rappresentazioni latenti

Una volta che abbiamo questi modelli piatti a nostra disposizione, possiamo invocarli per l’addestramento.

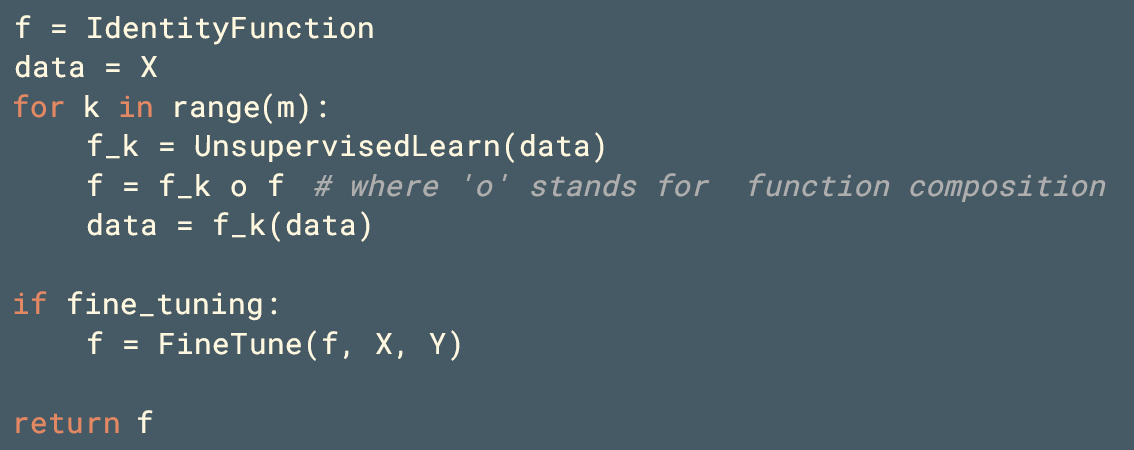
Si costruisce una rete profonda impilando uno sull’altro tanti modelli “piatti” fino a quando si arriva fino all’output. Ogni livello è **addestrato in modo** **indipendente** (rispetto a tutti gli altri) a partire dal proprio input per costruire un nuovo output. Il livello successivo è addestrato in modo indipendente rispetto a tutti gli altri prendendo come input l’output del livello precedente cercando di ricostruirlo.

Pensandolo con gli auto-encoders:



Ogni livello viene riqualificato utilizzando l'apprendimento non supervisionato, prendendo l'output del livello precedente e producendo come output una nuova rappresentazione dei dati, la cui distribuzione è, si spera, più semplice.

# Algoritmo



f è la funzione identità

Si vuole modificare la variabile f in modo tale che contenga il modello finale addestrato completamente.

X matrice contenente gli esempi

m iterazioni (m è il numero di livelli):

* ad ogni iterazione costruisco un nuovo modello f(k) utilizzando un apprendimento non supervisionato. Ad esempio si addestra un auto-encoder.
* f(k) è una funzione che parte dal suo input e restituisce la rappresentazione h.
* Dopodiché si calcola la nuova f mediante la composizione funzionale tra f(k) e f → f(x) = fk(f(x)).
* Cambio i dati in modo tale che al passo successivo, si apprende a partire dalla nuova rappresentazione h appena calcolata
* Al passo successivo, la f(k+1) prende i dati della f(k).

Si ottiene la seguente rete formata da un livello di input, poi un certo numero di livelli pre-addestrati in modo non supervisionato ed infine un livello di output.

A questo punto, si potrebbe applicare il **fine tuning** che consiste nell’addestrare in maniera supervisionata l’intera rete (che era stata inizializzata con i pesi appresi in modo non supervisionato) con gli esempi etichettati a disposizione.

Il fine tuning è facoltativo.

Motivi per cui il **fine tuning** è facoltativo:

1. Se si volesse creare semplicemente un auto-encoder profondo usando questa tecnica non faccio fine tuning.
2. Effettuare il pre-training di tutti i livelli con questa tecnica e poi lasciare all’utente di fare il fine tuning. Non sempre viene fatta nella funzione, si lascia libertà all’utente di farla quando vuole.

# Note

Questa procedura ha giocato un ruolo importante nel 2006, quando è stato dimostrato che poteva essere utilizzata per trovare buoni punti di inizializzazione per l'addestramento congiunto di architetture profonde.

Oggi sappiamo che per addestrare con successo (anche) l'architettura completamente connessa non è richiesto un pre-addestramento, ma questo approccio è stato il primo metodo a dimostrare che era possibile.

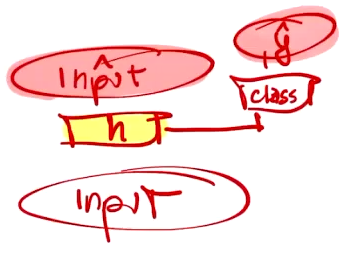
# Caratteristiche

* è un algoritmo **Greedy**: non c’è alcun tentativo di arrivare a un ottimo globale.
* è un **Layer-Wise**: si procede livello per livello
* i livelli sono addestrati in modo **non supervisionato (Unsupervised)**
* **Pre-Training**: si suppone che questo sia soltanto il primo step necessario prima di far partire il fine tuning. Passo iniziale per inizializzare i pesi di una rete profonda supervisionata.

# Greedy Layer-Wise Supervised Pre-Training

L’algoritmo può essere anche usato come passo iniziale per modelli non supervisionati profondi, in cui l’obiettivo è imparare qualcosa sulla struttura dei dati. Caso più ovvio: apprendimento di un auto-encoder profondo.

E’ anche possibile avere **Greedy Layer-Wise Supervised Pre-training**. Ogni modello può essere pensato nel seguente modo: input → rappresentazione h → classificatore → stima delle etichette (funzione di loss che mette insieme errore di ricostruzione e errore sulle etichette). Output (ricostruzione input). Questo è un modo interessante in quanto le etichette guidano la ricostruzione della rappresentazione h.



È comune usare la parola “pre-training” per riferirsi non solo alla fase di pre-training stessa, ma all'intero protocollo in due fasi che combina la fase di pre-training e una fase di apprendimento supervisionato. **La fase di apprendimento supervisionato può comportare la formazione di un semplice classificatore in aggiunta alle caratteristiche apprese nella fase di pre-addestramento** (**Supervised Pre-Training**), oppure può coinvolgere la messa a punto supervisionata dell'intera rete appresa nella fase di pre-addestramento (Unsupervised Pre-Training).

# Perchè funziona Unsupervised Pre-Training?

Il pretraining non supervisionato non funziona sempre. Esistono alcuni studi che dimostrano che questa tecnica potrebbe in alcuni casi peggiorare le performance della rete.

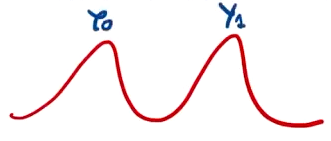
Si tratta di una tecnica unsupervised: è possibile che, apprendendo la struttura dei dati, ci si allontana da una buona rappresentazione per il task specifico che si sta risolvendo. Quando la struttura dei dati non permette di predire facilmente l’etichetta, si potrebbe andare verso una direzione peggiore.

La tecnica Unsupervised Pre-Training è usata con un modo di inizializzare una rete profonda supervisionata.

Data una rete profonda, si vuole imparare dei buoni punti di inizializzazione per i parametri nei vari livelli. Si vuole inizializzare i parametri non a partire da valori piccoli, ma metterli in un punto dove si riesce a fare discesa del gradiente più facilmente.

Due principali idee sono importanti per capire perché e quando funziona il pre-training non supervisionato:

1. La **scelta dei parametri iniziali** di questi modelli può avere effetti di regolarizzazione importante sul modello.
2. **Imparare la struttura dei dati in input** (distribuzione) può aiutare ad apprendere il mapping dall’input all’output. Se imparo la curva rossa, poi è più semplice predire y0 e y1.



## Scelta dei parametri iniziali

Per tanto tempo si è assunto che questa tecnica funzionasse perché questo modo di inizializzare i parametri portano a individuare minimi locali diversi da quelli che si andava a considerare nel momento in cui si inizializzano i parametri casualmente con valori piccoli.

Per tanto tempo si è pensato che il problema di apprendimento delle reti profonde fosse dovuto al fatto che la funzione di loss avesse tanti minimi locali, molto frastagliata e con tanti minimi locali anche molto diversi dal minimo globale.

Simile a questo modello.



In verità la funzione di loss è molto frastagliata e presenta tanti minimi locali ma questi minimi locali sono già molto buoni. Molto simile a questo modello.



Ci si sposta in un punto che ci permette di trovare un minimo locale particolarmente buono. Questo tipo di inizializzazione ci porta a considerare regioni dello spazio che naviga la rete che magari non avrei considerato se avessi inizializzato i pesi in modo casuale a valori piccoli, evidentemente perché queste zone sono zone in cui alcuni pesi sono molto grandi e altri molto piccoli (zone non omogenee che sono difficili da trovare inizializzando i parametri a caso).

## Apprendere la distribuzione dei dati in input

Apprendere qualcosa sulla distribuzione dei dati in input è molto importante per fare predizione correttamente in futuro.

L’idea è simile al Transfer Learning in cui ciò che si impara in un task supervisionato, si può applicare su un altro task supervisionato.

Idea: Ciò che si impara in modo non supervisionato si può utilizzare in un task supervisionato.

Applicando questa idea sull’apprendimento delle immagini, la rappresentazione appresa dalla rete convolutiva identifica delle features delle immagini in input che possono essere utili per fare classificazione.

Ad esempio: feature che si attivano quando ci sono delle ruote, persone… Questo lo si può fare in modo non supervisionato.

Poi si può applicare questa rappresentazione appresa per un task di classificazione, è molto più semplice.

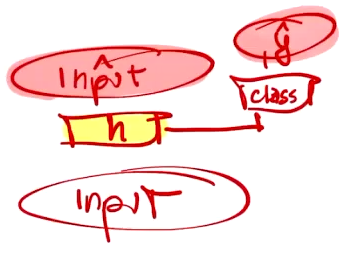
I dettagli di come imposto la rete possono essere importanti per ottenere benefici. Se si usano unità lineari, sto forzando la rete a imparare una rappresentazione interna che permette di separare linearmente etichette positive da quelle negative. Se uso unità non lineari, questa proprietà non ce l’ho.

L’architettura della rete può forzare la rappresentazione ad assumere proprietà che troviamo interessanti e che vogliamo.

# Apprendimento supervisionato e non supervisionato simultaneo

Non c’è nessuna garanzia che le feature che apprendiamo in modo supervisionato abbiano delle proprietà interessanti che vogliamo.

Utilizzare l’informazione anche delle etichette sin da subito può vincolare la rappresentazione ad andare verso una direzione che ci aiuti per il task che dobbiamo rappresentare. Questo funziona molto bene.



# Quand'è che questi tipi di approcci funzionano bene?

**Quando la rappresentazione originale è povera**: la rappresentazione dei dati in input è tale per cui è difficile cogliere una struttura da questi dati. Apprendere una rappresentazione più ricca dei dati può aiutare molto.

Un esempio è l’apprendimento del word embedding nel linguaggio naturale.

[Il **word embedding** (rappresentazione distribuita delle parole) permette di memorizzare le informazioni sia semantiche che sintattiche delle parole partendo da un corpus non annotato e costruendo uno spazio vettoriale in cui i vettori delle parole sono più vicini se le parole occorrono negli stessi contesti linguistici, cioè se sono riconosciute come semanticamente più simili.]

Le parole sono rappresentate da un vettore one-hot e sono particolarmente povere.

La parola casa è rappresentata dal vettore (000…..10…..0)

La parola cane è rappresentata dal vettore (0….10.……..0)

La parola gatto è rappresentata dal vettore (0…………1..0)

‘gatto’ è molto simile alla parola ‘cane’ ma analizzando la distanza dei vettori tutti e tre i vettori sono alla stessa distanza, differiscono di una sola posizione. Queste distanze non ci dicono nulla sulla semantica che stiamo guardando.

Nell’apprendimento di word embedding si impara proprio a partire da questa rappresentazione e arrivare ad una rappresentazione con annotazione semantica → la distanza fra casa e gatto è molto più alta rispetto alla distanza fra cane e gatto.

**Quando la rappresentazione è povera, è importante apprendere una rappresentazione anche in modo non supervisionato.**

Vedere il pre-training non supervisionato come un regolarizzatore, questo passo è molto importante **quando il numero di esempi è piccolo** perché come tutti i regolarizzatori, il regolarizzatore è efficace quando abbiamo pochi dati rispetto al numero di parametri da apprendere.

Con un numero infinito di dati non abbiamo bisogno di regolarizzare.

Per la stessa ragione, è anche molto importante **quando il numero di esempi non etichettati è grande**. Se il numero di esempi non etichettati è grande, posso usare queste tecniche per apprendere la struttura dei dati da questi esempi. Questa inizializzazione dei parametri della rete che faccio a partire dagli esempi non etichettati, agisce come regolarizzatore nel momento in cui si fa fine tuning della rete utilizzando anche i dati etichettati.

Un altro momento in cui è molto utile il pre-training non supervisionato è **quando** **la funzione da apprendere è estremamente complicata**. Questo perché in genere avrà una **varianza dell’errore maggiore** quindi si avrà maggiormente bisogno di un regolarizzatore per la rete.

E’ interessante notare che a differenza di altri regolarizzatori, questo tipo di regolarizzatore **non forza la funzione che dobbiamo apprendere a essere semplice**. Può essere un modo migliore rispetto ad altri regolarizzatori per apprendere una funzione complicata.

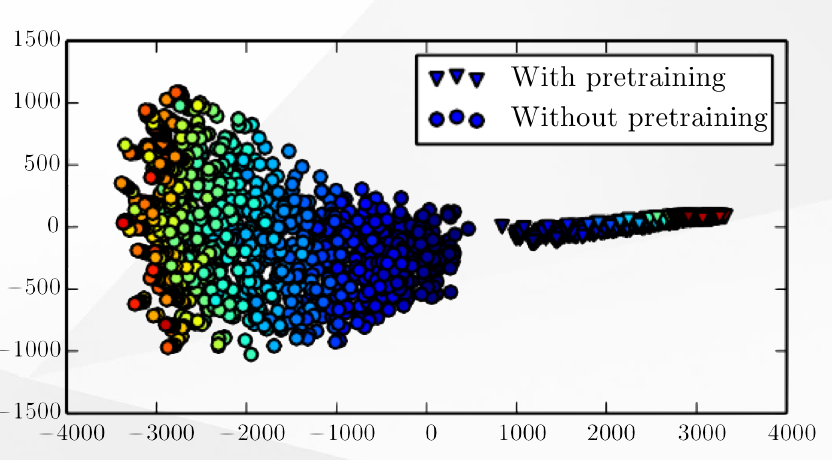
Invece di forzare la funzione a essere semplice, questo tipo di regolarizzatore **aiuta a scoprire le features dei dati che sono maggiormente utili per risolvere il problema** e questo lo fa quando il **task** è di tipo **non supervisionato**.

Per concludere, **se la funzione da apprendere è complicata e la struttura dei dati determina le etichette che voglio assegnare agli esempi, allora il pre-training non supervisionato è molto utile.**

# Il pre-training riduce la varianza dell’errore

L’errore di classificazione può essere decomposto come la somma di tre quantità: errore di polarizzazione, errore di varianza, errore dovuto al rumore.

L’errore di varianza si ha quando l’algoritmo di apprendimento tende ad apprendere classificatori molto diversi a fronte di pochissime variazioni dei dati, ovvero quando il rapporto tra quanto cambiano i dati e quanto cambia la funzione che apprendo è sbilanciato in modo tale che apprendo cose molto diverse a fronte di piccole variazioni dei dati.



Nel grafico, ogni punto/triangolo è una rete neurale.

I punti rappresentano reti neurali apprese senza pre-training.

Invece i triangoli rappresentano reti neurali apprese con pre-training.

Si nota che la varianza nelle reti apprese nel secondo caso sia molto più piccola rispetto alla varianza ottenuta da reti apprese senza pre-training.

La fase di pre-training inizializza i pesi per modellare la struttura dei dati e una volta modellata, si fa solo fine tuning, quindi si fanno piccole variazioni per riuscire a fare la parte di classificazione correttamente.

Quindi se non si fa pre-training ottengo molta più variazione.

# Svantaggi

Come tecnica di regolarizzazione, il pre-training non supervisionato ha il problema che è **difficile da calibrare**.

Esistono molti modi per fare regolarizzazione.

La maggior parte delle volte si ha qualche parametro (ad esempio lambda ƛ) che regola quanto è l’entità di regolarizzazione che si vuole apportare.

La maggior parte delle volte si ha una funzione loss che confronta le etichette con le etichette target e poi un qualche funzione della norma dei pesi ||w||^2. Associato al termine di regolarizzazione, spesso è presente il termine lambda ƛ che indica quanto è importante deve essere la regolarizzazione.

Loss(y^, y) + ƛ ||w||^2.

Il termine lambda rende semplice calibrare la regolarizzazione in modo tale da non regolarizzare troppo ma nemmeno troppo poco.

Usando invece il pre-training non supervisionato come tecnica di regolarizzazione, questo non può essere fatto: o si fa addestramento oppure non si fa. Quindi non abbiamo il termine lambda ƛ per calibrare la regolarizzazione.

L’unica cosa che si può fare è **cambiare l’architettura della rete**, quindi gli **iperparametri** che determinano le varie reti shallow che si usano per la fase di pre-training. Questi iperparametri in genere sono spesso complessi e difficili da calibrare e soprattutto che cambia un pezzettino prima di capire cosa succede, bisogna fare una fase di pre-training su tutti i livelli quindi molto complicato e molto lenta.

# Come viene usata oggi questa tecnica

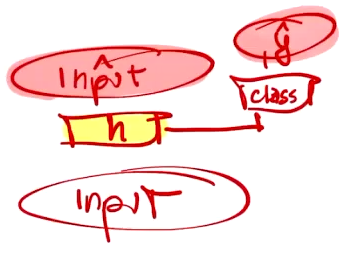
La tecnica di apprendimento greedy non supervisionato layer-wise pre-training è stata per lo più abbandonata.

Un’eccezione rilevante è nel campo del **processamento del linguaggio naturale**, dove la rappresentazione delle parole tramite vettori one-hot è talmente brutta e quindi è necessaria una tecnica per migliorare questa rappresentazione. Gran parte della ricerca del linguaggio naturale si dedica al word embedding.

Altre **tecniche di regolarizzazione** per il deep learning, come il **dropout** o **batch normalization funzionano meglio** rispetto al pre-training non supervisionato.

Mentre il pre-training supervisionato è invece molto popolare, funziona molto bene.

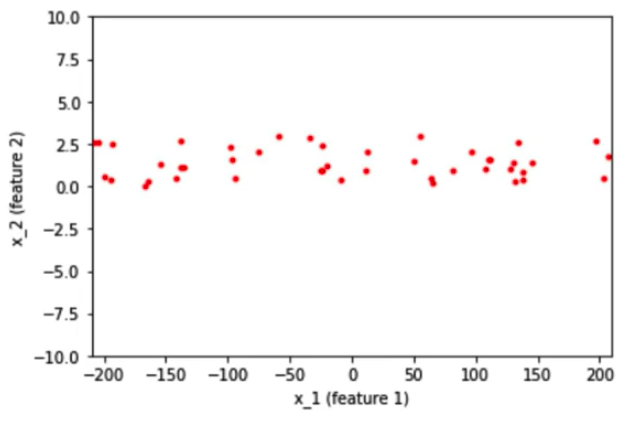
La versione supervisionata è questa:



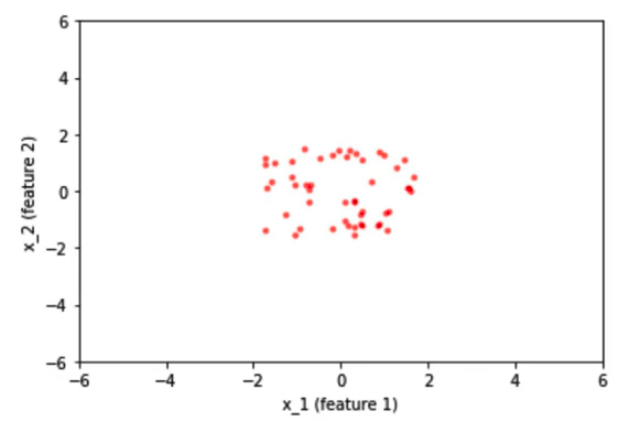
Dropout e Batch normalization

# Perché è necessario normalizzare gli input?

Si consideri uno scenario in cui abbiamo dati 2D con le features x\_1 e x\_2 che entrano in una rete neurale. Una di queste feature x\_1 ha uno spread più ampio da -200 a 200 e un'altra feature x\_2 ha uno spread più ristretto da -10 a 10.



Una volta normalizzati i dati, la diffusione dei dati per entrambe le caratteristiche è concentrata in una regione, cioè… da -2 a 2. La diffusione sarebbe simile a questa.



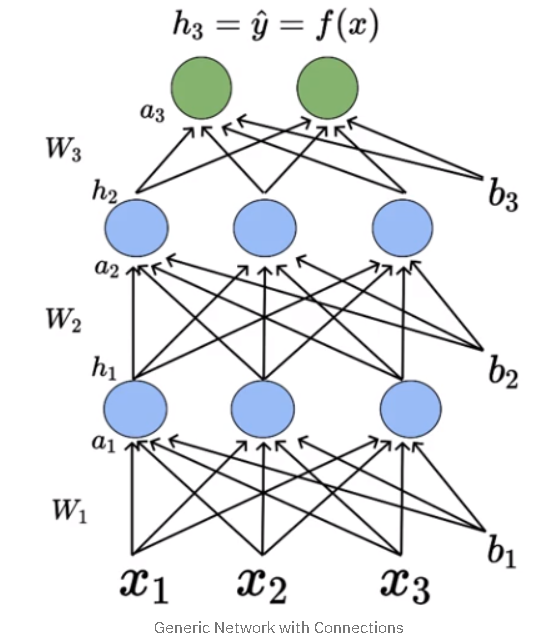
Prima di normalizzare gli input, i pesi associati a questi input varierebbero molto perché le features degli input presenti in intervalli diversi variano da -200 a 200 e da -2 a 2. Per accogliere questa differenza di intervallo tra le features che alcuni pesi dovrebbero essere grandi e poi alcuni devono essere piccoli. Se disponiamo di pesi maggiori, anche gli aggiornamenti associati alla backpropagation sarebbero grandi e viceversa. A causa di questa distribuzione non uniforme dei pesi per gli input, l'algoritmo di apprendimento continua a oscillare nella regione di plateau prima di trovare i minimi globali.

Per evitare che l'algoritmo di apprendimento trascorra molto tempo oscillando nel plateau, normalizziamo le features di input in modo che tutte le caratteristiche siano sulla stessa scala. Poiché i nostri input sono sulla stessa scala, anche i pesi ad essi associati sarebbero sulla stessa scala. Aiutando così la rete ad allenarsi più velocemente.

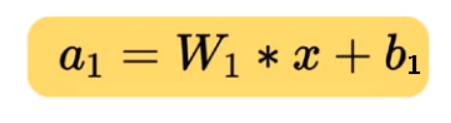
# Batch Normalization

**Abbiamo normalizzato gli input, ma per quanto riguarda i rappresentanti nascosti?**

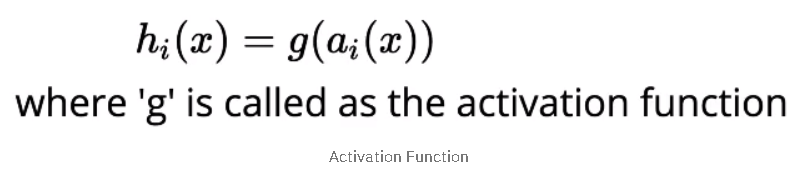
Normalizzando gli input siamo in grado di portare tutte le caratteristiche degli input sulla stessa scala. Nella rete neurale, dobbiamo calcolare la pre-attivazione per il primo neurone del primo strato a₁₁. Sappiamo che la pre-attivazione non è altro che la somma ponderata degli input più il bias. In altre parole, è il prodotto scalare tra la prima riga della matrice dei pesi W₁ e la matrice di input X più la polarizzazione b₁₁.



L'equazione matematica per la pre-attivazione in ogni strato "i" è data dalla seguente equazione



L'attivazione ad ogni strato è uguale ad applicare la funzione di attivazione all'uscita della pre-attivazione di quello strato. L'equazione matematica per l'attivazione a ogni livello "i" è data da

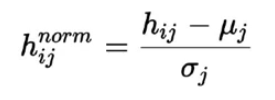


Allo stesso modo, è necessario calcolare i valori di attivazione per il numero "n" di livelli nascosti presenti nella rete. I valori di attivazione fungeranno da input per i successivi livelli nascosti presenti nella rete. Quindi non importa cosa abbiamo fatto all'input, indipendentemente dal fatto che li abbiamo normalizzati o meno, i valori di attivazione varierebbero molto man mano che andiamo sempre più in profondità nella rete in base al peso associato al neurone corrispondente.10.2 Explicit density models

Per portare tutti i valori di attivazione sulla stessa scala, normalizziamo i valori di attivazione in modo tale che la rappresentazione nascosta non vari drasticamente e ci aiuti anche a ottenere un miglioramento nella velocità di allenamento.

**Perché si chiama batch normalization?**

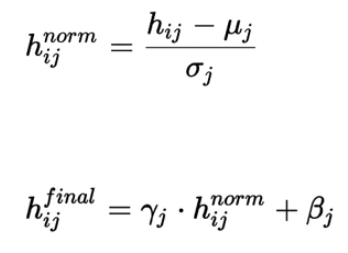
Poiché stiamo calcolando la media e la deviazione standard da un singolo batch invece di calcolarle dall'intero dato. La normalizzazione in batch viene eseguita individualmente su ciascun neurone nascosto nella rete.



**Apprendere gamma e beta**

Poiché stiamo normalizzando tutte le attivazioni nella rete, stiamo applicando alcuni vincoli che potrebbero deteriorare le prestazioni della rete?

Al fine di mantenere la potenza rappresentativa della rete neurale nascosta, la normalizzazione batch introduce due parametri aggiuntivi: Gamma e Beta. Una volta normalizzata l'attivazione, è necessario eseguire un ulteriore passaggio per ottenere il valore di attivazione finale che può essere fornito come input per un altro livello.

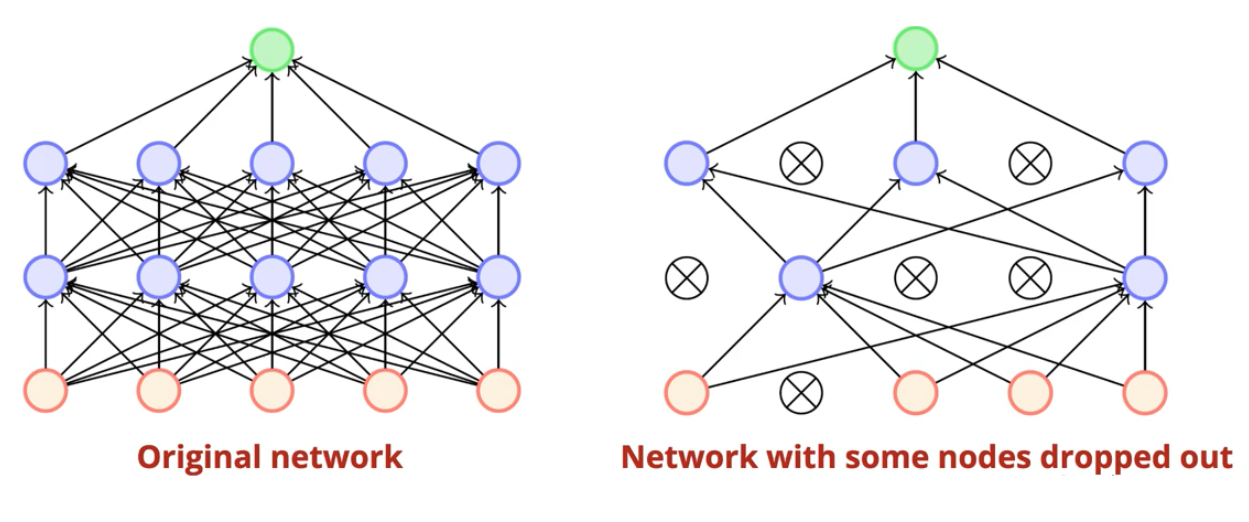


I parametri Gamma e Beta vengono appresi insieme ad altri parametri della rete. Se Gamma (γ) è uguale alla media (μ) e Beta (β) è uguale alla deviazione standard (σ) allora l'attivazione h\_final è uguale a h\_norm, preservando così la potenza rappresentativa della rete.

# Dropout

Il dropout è una tecnica di regolarizzazione che “elimina” o “disattiva” casualmente alcuni neuroni nella rete neurale al fine di evitare il problema dell'overfitting.

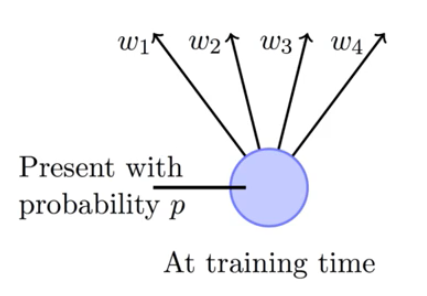
L'addestramento di una rete neurale profonda con parametri di grandi dimensioni sui dati potrebbe portare a un overfitting. Possiamo addestrare più reti neurali con diverse configurazioni sullo stesso set di dati e prendere il valore medio di queste previsioni?



Creare un insieme di reti neurali con architetture differenti e addestrarle non sarebbe fattibile nella pratica. Dropout in soccorso.

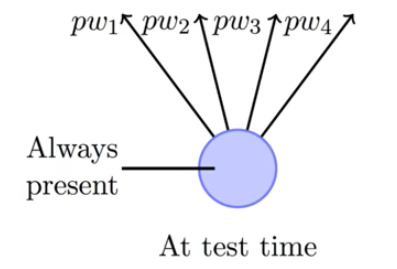
Il dropout disattiva i neuroni in modo casuale ad ogni fase di addestramento invece di addestrare i dati sulla rete originale, addestriamo i dati sulla rete con i nodi esclusi. Nella successiva iterazione della fase di addestramento, i neuroni nascosti che vengono disattivati ​​dal dropout cambiano a causa del loro comportamento probabilistico. In questo modo, applicando dropout, cioè ... disattivando casualmente alcuni nodi individuali durante l'addestramento, possiamo simulare un insieme di reti neurali con architetture differenti.

**Dropout al tempo di training**



In ogni iterazione di addestramento, ad ogni nodo della rete è associata una probabilità p se rimanere nella rete o disattivarlo (dropout) fuori dalla rete con probabilità 1-p. Ciò significa che i pesi associati ai nodi sono stati aggiornati solo p frazione di volte perché i nodi sono attivi solo p volte durante l'addestramento.

**Dropout al tempo di test**



Durante il tempo di test, consideriamo la rete neurale originale con tutte le attivazioni presenti e ridimensioniamo l'output di ciascun nodo di un valore p. Poiché ogni nodo viene attivato le uniche p volte.

**Appunti Esposito**

Sono due metodi di normalizzazione delle reti neurali fondamentali per apprendere correttamente i modelli profondi.

**Dropout**

Si considerino due livelli fully-connected. Uno dei problemi che può succedere è che il livello in alto impari a fare molto affidamento su poche features del livello in basso. Il livello in basso in realtà impara delle features che alla fine vengono buttate e si fa riferimento solo a poche features.

Svantaggi:

* spreco di potenza → il livello in basso ha molte più informazioni da trasmettere, ma conta solo su poche features
* rende questo modello molto fragile → nel momento in cui le due features stanno dicendo una cosa sbagliata, tutti i livelli al di sopra sbagliano completamente.



Durante l’apprendimento, ogni volta che si fa un passo forward, si disattiva una certa percentuale dei neuroni ponendoli a 0 (in genere 50%). Il livello non può più fare affidamento a pochi neuroni del livello sottostante perché in molti casi quei neuroni non dicono più nulla perché soppressi.

Al termine dell’apprendimento, si toglie il dropout e si fa fluire tutta l’informazione.

Questa tecnica permette di ottenere delle reti che apprendono meglio.

**Batch normalization**

Spesso nelle reti neurali si normalizza il dataset prima di iniziare l’apprendimento: si toglie la media, la si divide per la deviazione standard. Questo perché aiuta i neuroni dei livelli successivi ad apprendere meglio.

**Batch normalization** è un’estensione di questa tecnica: anziché applicarla solo sul dataset iniziale, la si applica su tutti i batch e su tutti i livelli (non solo sul livello di input). Si normalizzano gli output da un livello ad un altro. Il modo in cui lo si fa non è prestabilito ma lo impara la rete. Si aggiungono due parametri per ogni neurone che indicano cosa bisogna sottrarre al neurone e per quanto lo devo dividere.

Anche questa tecnica funziona molto bene.

Dropout e batch normalization permettono di apprendere le reti profonde senza usare la tecnica di apprendimento non supervisionato. Invece la tecnica di apprendimento supervisionato funziona molto bene.

Transfer Learning and Domain Adaptation

Transfer Learning e Domain Adaptation sono tecniche che fanno riferimento a una situazione in cui si apprende qualcosa in un certo ambiente (distribuzione P1) e si usa questa informazione appresa sulla distribuzione per inferire qualcosa su un nuovo ambiente (distribuzione P2).

P1 e P2 sono connesse fra loro.

E’ una **generalizzazione dell’idea del greedy pre-training**: è la stessa idea ma P1 e P2 sono ben delineate, P1 è il contesto in cui si fa apprendimento non supervisionato, P2 in cui si fa il passaggio all’apprendimento supervisionato.

# Transfer Learning: caso generale

Nel Transfer Learning, l’algoritmo di apprendimento è di fronte a due o più obiettivi di apprendimento. Si assume che molti fattori che spiegano variazioni nella distribuzione P1 sono rilevanti alle variazioni che devono essere catturate per apprendere P2.

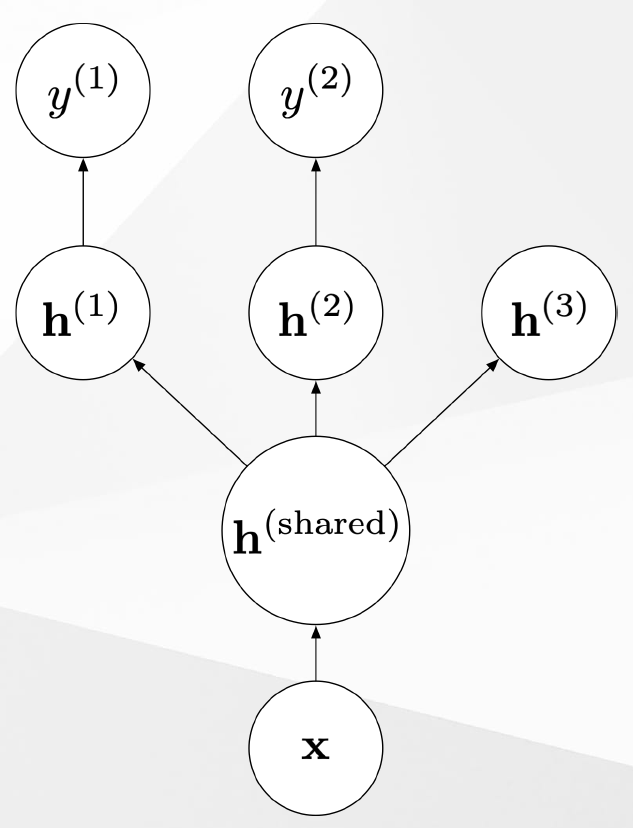
Quando parliamo dei “fattori che spiegano variazioni in P1” si fa riferimento a caratteristiche che permettono di apprendere P1. In una situazione di transfer learning, **si assume che P1 e P2 abbiano fattori in comune che spieghino sia P1 che P2**.

L’input è lo stesso oppure ha la stessa forma. **Ciò che cambia è il modo di etichettare i dati.**

Un esempio: apprendere come sono classificate immagini etichettate con due insiemi di etichette differenti (non necessariamente le stesse immagini ma il loro formato/descrizione è la stessa).

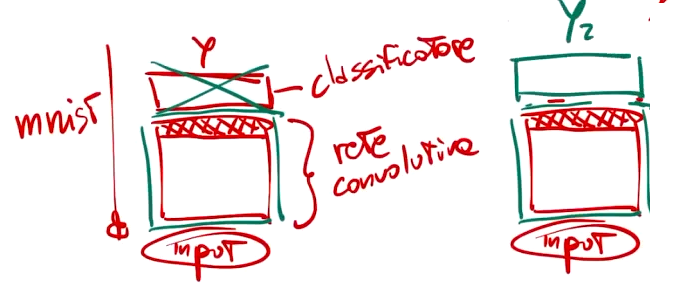
In uno studio, è stato fatto transfer learning imparando su **MNIST** (immagini in b/n di cifre scritte a mano da utente), poi sono stati usati i pesi appresi dalla parte convolutiva per addestrare un classificatore che fa predizione al **cancro alla mammella**. Le immagini in b/n con caratteristiche molto diverse dai numeri scritti a mano da un utente ma le caratteristiche apprese dalla rete convolutiva per distinguere i numeri si dimostrano utilissime per analizzare le immagini della mammografia.

Quando si parla di transfer learning, nella maggior parte dei casi, siamo di fronte a un modello in cui dall’input x ci interessa creare una rappresentazione h condivisa fra più task di apprendimento. Questa rappresentazione condivisa è specializzata in h1, h2, h3 per risolvere i task veri e propri.



Cosa succede quando si fa transfer learning?

Sia una rete neurale convolutiva composta da una parte di rappresentazione. L’ultimo livello di rappresentazione è la nuova rappresentazione che viene accoppiata con un classificatore utile per predire y.

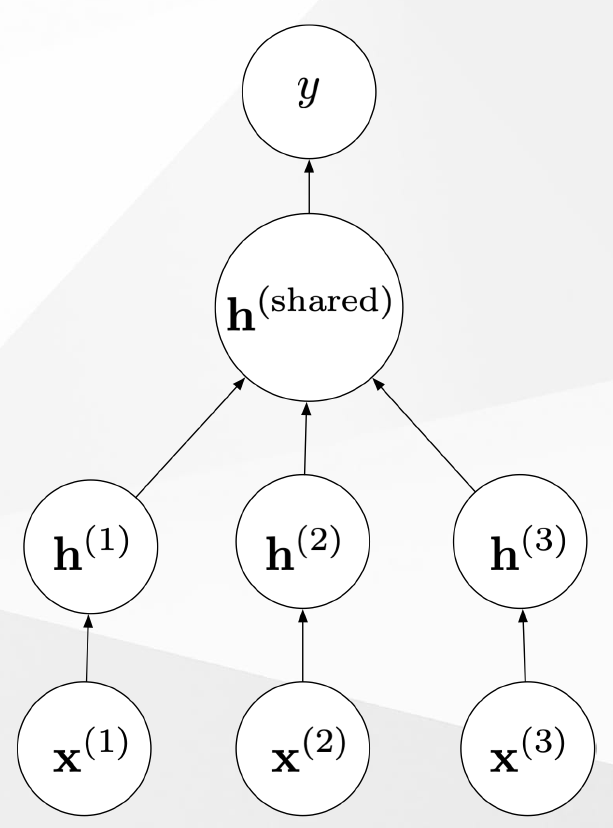


Quando si fa transfer learning, si addestra tutta questa rete su MNIST. Dopodiché il classificatore viene buttato via. Invece tutta la rete convolutiva viene copiata e immessa in un nuovo sistema a cui si aggiunge un ulteriore classificatore che predice il nuovo target (cancro alla mammella).

Tra le due reti, ciò che cambia è la parte che costruisce il classificatore (nello schema del Transfer Learning corrisponde a h1, h2…).

# Transfer Learning: Caso particolare

In alcune situazioni la rappresentazione condivisa è vicina all’output. Bisogna creare delle rappresentazioni di raccordo tra i punti diversi e il modello condiviso.



Questo nuovo modello è utile quando gli input provengono dalla stessa distribuzione ma sono un po’ diversi gli uni dagli altri e c’è bisogno di eliminare dagli input ciò che li rendono diversi.

x1,x2,x3 hanno rappresentazioni h1,h2,h3 ma un po’ diversi l’uno dall’altra. La rappresentazione condivisa viene costruita facendo una sintesi delle diverse rappresentazioni h1,h2,h3.

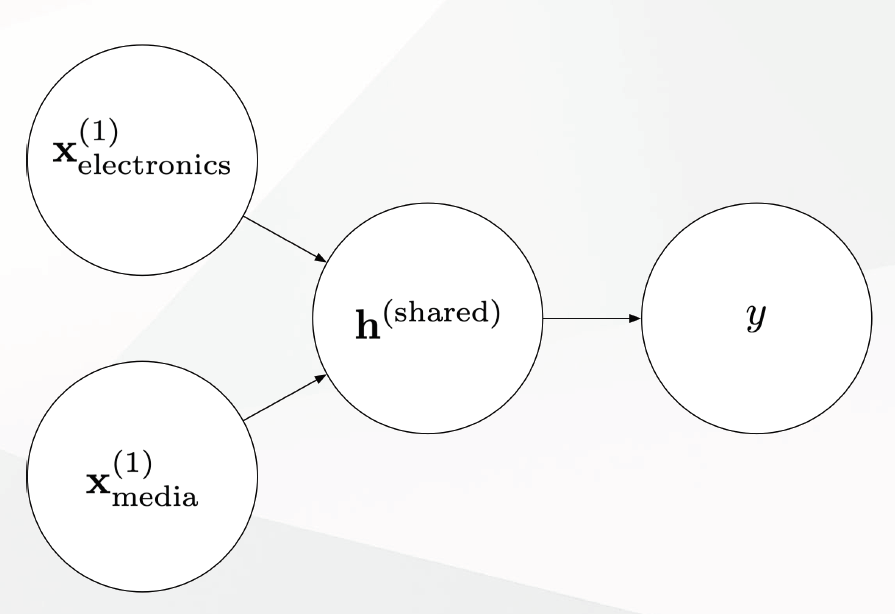
Esempio tipico è il **riconoscimento del parlato**. Quando impariamo a riconoscere il parlato, la rappresentazione alla base del riconoscimento è la stessa per tutte: distinguere parole, fonemi… ma ogni parlante ha una pronuncia leggermente diversa. Se tentassimo di fare direttamente il salto dal parlante x1 a y e dal parlante x2 a y avremo dei problemi perché i due parlanti pronunciano in modo leggermente diverso.

Per evitare questo problema si ha una rappresentazione condivisa h delle parole e fonemi… ma un punto intermedio permette di adattare le differenze specifiche dei parlanti al modello condiviso.

# Domain Adaptation

Nel Transformation Learning abbiamo un cambio di obiettivo da un task all’altro (da predire cifre numeriche a predire cancro alla mammella).

Nel **Domain Adaptation** il task rimane lo stesso, ma le distribuzioni in input sono diverse.



Esempio: **analisi dei sentimenti** (estrarre dal testo il sentimento dell’autore).

In genere si vuole applicare lo stesso tipo di analisi a contesti diversi (es. tweet su twitter oppure revisioni dei prodotti su un sito e-commerce).

# Concept Drift

Un altro contesto in cui è utile l’idea di Transfer Learning e Domain Adaptation è il **Concept Drift**.

Il concept drift è un termine usato in apprendimento automatico per indicare quando il concetto che sottende un certo task si sposta.

Ad esempio, si sta cercando di prevedere l’interesse medio dei mutui in una certa zona e costruisco un modello sui dati. Poi c’è un cambiamento delle condizioni generali dell’economia per cui tutte le variabili apprese in quel modello non vanno più bene perché l’interesse medio è cambiato totalmente in quanto sono cambiate tutte le condizioni attorno.

In questo caso il transfer learning può aiutare molto perché non devo apprendere da zero tutto il modello, ma posso partire da una rete che è stata pesata sul vecchio modello che coglie bene le caratteristiche importanti di quel task e adattarla alla nuova distribuzione dei dati che sta giungendo.

<https://translate.google.com/translate?hl=it&sl=en&u=https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/&prev=search&pto=aue>

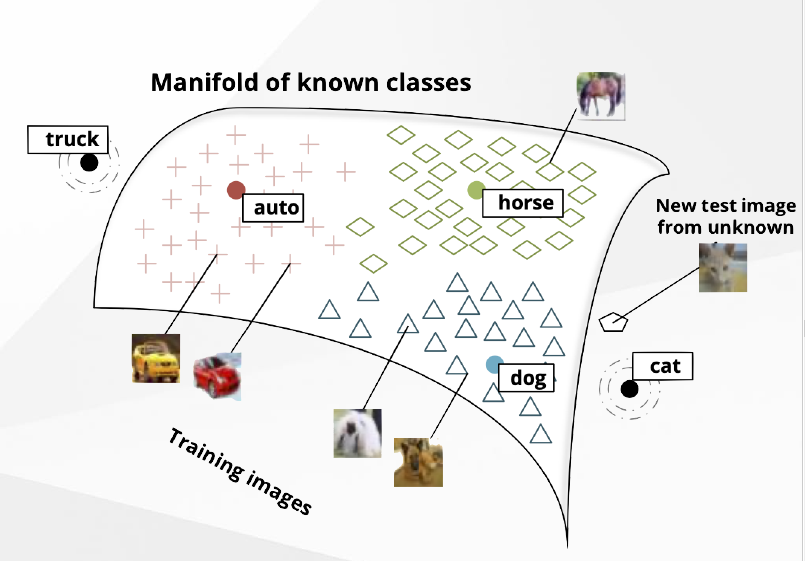
# One-Shot Learning and Zero-Shot Learning

Casi estremi di transfer learning sono gli One Shot Learning e lo Zero Shot Learning.

## One Shot Learning

Cercare di imparare da un singolo esempio etichettato. Si può immaginare di avere una moltitudine di esempi ma abbiamo solo un esempio con etichetta. Funziona bene quando i dati sono distribuiti in uno spazio latente in modo tale da separare bene le classi.

E’ importante apprendere bene una **rappresentazione latente** che sia adatta a questo tipo di task perché nel momento in cui arriva una nuova immagine associata ad una nuova etichetta (gatto), se cade in un punto lontano rispetto a tutte le altre etichette, allora si può prevedere che tutta la zona circostante è relativa all’etichetta “cat”. Quindi questo permette di generalizzare da una sola immagine.



Uno dei problemi delle reti neurali è che hanno bisogno di tantissimi dati per lavorare. Questo è un modo per venirne fuori usando dati non etichettati. Non si sta parlando di un dataset con 4 esempi in tutto ma di un dataset con tanti esempi di cui solo 4 etichettati.

## Zero-Shot Learning

L’evoluzione più estrema del One-Shot Learning è lo Zero-Shot Learning, detto anche zero-data learning.

Ci dobbiamo adattare a una nuova situazione senza avere alcun esempio etichettato da cui trarre informazioni relative all’etichetta.

Ovviamente questo non può portare da nessuna parte se non ho alcuna informazione che mi permette di ricostruire l’etichetta. Si può riformulare il task di zero-shot learning con un task che include 3 variabili random:

* x : inputs
* y : outputs
* T : descrizione del task

Si vuole addestrare un modello basato sulla probabilità *p(y|x,T)*.

Un esempio di questo tipo di approccio: task di **image recognition** in cui la descrizione T è la descrizione dell’oggetto da riconoscere. Oltre alle immagini, ho un testo descrittivo dell’immagine (esempio: y=1 se c’è un gatto nella scena e T descrive i gatti come aventi 4 gambe e orecchie a punta).

Non ho alcun esempio etichettato con l’etichetta “gatto” ma se viene sottoposta un’immagine con queste caratteristiche si può classificare con quell’etichetta.

Semi-Supervised Learning e Causalità

**Con il termine “causalità” si intende identificare le cause che determinano le etichette nei dati**.

**Cos’è che rende una rappresentazione migliore rispetto ad un’altra?**

Ipotesi: **la rappresentazione ideale è quella che riesce a determinare quali sono le cause sottostanti ai dati osservati**. Se la rappresentazione riesce a identificare queste cause allora si può usare questa rappresentazione per predire le etichette.

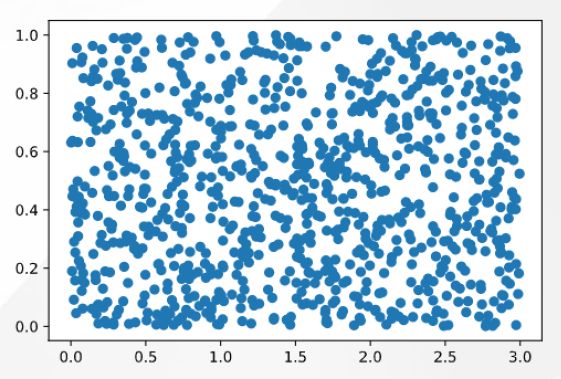
L’ipotesi della causalità sottolinea una grande quantità di ricerca motivata dall’idea che districare i fattori che causano la distribuzione p(x) potrebbe essere un buon passo per l'apprendimento di p(y|x).

Nota: una rappresentazione h che districa le cause sottostanti di x non necessariamente è semplice da modellare come quelle dove le features sono sparse o indipendenti.

# SSL e causalità: un fallimento

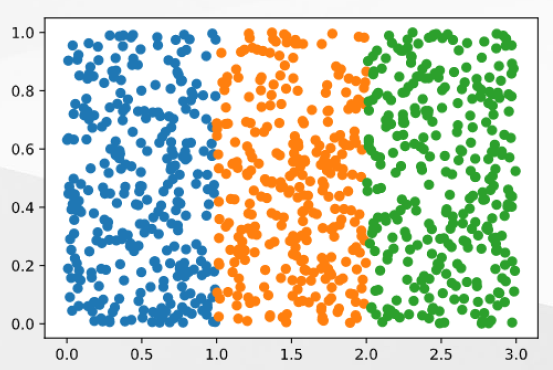
Sfruttando una rappresentazione per p(x), l’apprendimento fallisce perché nell’esempio si assume che p(x) è una distribuzione uniforme.

Quando vado a fare la parte non supervisionata, quindi cerco di imparare la distribuzione dei dati e osservo il seguente plot, non esiste alcuna struttura che posso sfruttare in alcun modo.



La Fig.1 porta pochissima informazione nel caso della Fig.2

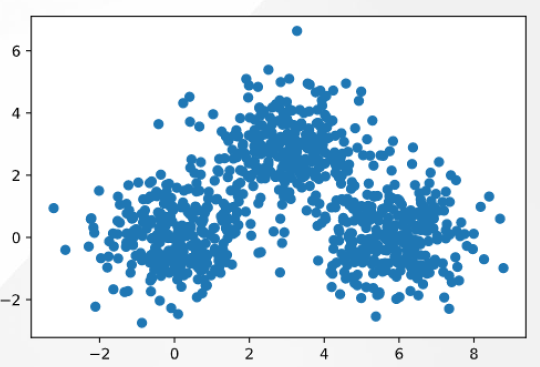
Successivamente, se osservo che le etichette sono messe nel seguente modo, posso imparare effettivamente come etichettarle.



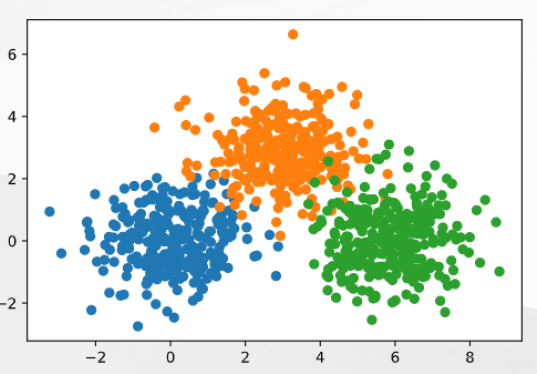
[A volte l’apprendimento semi-supervisionato fallisce perché l’apprendimento non supervisionato di p(x) non fornisce alcun aiuto ad apprendere p(y|x).]

# SSL e causalità: un successo

Se si apprende a partire da una rappresentazione di questo tipo:



ovvero si cerca di modellare p(x) e si osserva questa struttura a cluster, può essere molto importante. Ho informazione a sufficienza per fare un’ottima predizione.



[Altre volte, la rappresentazione y è tra le cause salienti di p(x). In questi casi, l’apprendimento p(x) può essere molto utile.]

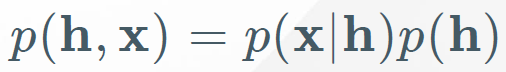
**Apprendere la struttura dei dati può essere utile**. La causalità in questo caso avviene nel seguente modo: nel momento in cui sono stati generati i dati si estrae un’etichetta (centroide) e poi si assegna agli esempi vicino l’etichetta corrispondente.

Aver estratto l’etichetta arancione ha determinato come ha causato l’estrazione dell’esempio nel piano bidimensionale.

Un altro esempio: è la stessa idea di quando ho 3 urne da cui estrarre una pallina. Le urne contengono distribuzioni delle palline diverse tra loro. Estraggo il numero dell’urna e poi estraggo la pallina dall’urna corrispondente. Il numero dell’urna è fattore causale che poi determina come saranno i fatti i dati che estrarrò successivamente.

# SSL e Causalità

Se h rappresenta tutti i fattori che causano x e si assume che y è collegato ad almeno uno di essi, allora il processo generativo della distribuzione congiunta p(h,x) è un modello dove prima si estrae h e poi si estrae x dato h.



Sotto questa ipotesi, la probabilità marginale di x viene calcolata come la media su tutti gli h della probabilità di x dato h.



formula della media nel caso finito, valore medio di probabilità di x dato h.

Se si vuole marginalizzare rispetto alla variabile h, bisogna sommare p(h,x) su tutte le possibili h.

Vogliamo marginalizzare rispetto alla variabile h per ottenere la p(x).

E\_h: media su tutti gli h

Ricostruire p(x) può essere molto utile per scoprire h e predire y.

Si sta cercando di trovare una rappresentazione h che riesca a modellare le cause che generano i dati. Per arrivare a modellare h, si cerca di modellare la p(x) perché la p(x) è collegata alle cause



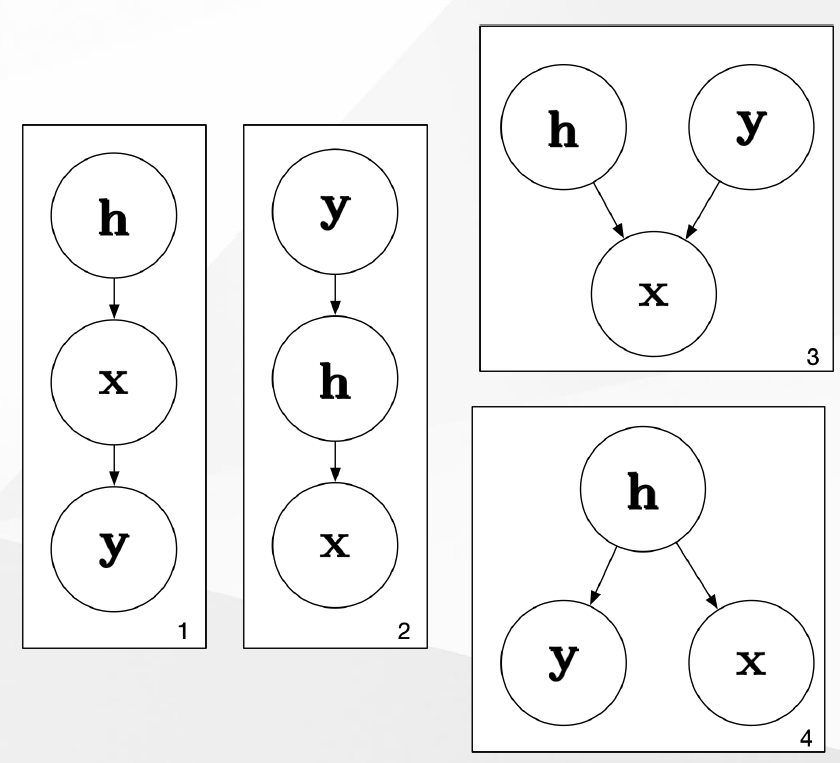
e quindi nel modellare questa probabilità, estrarrò qualcosa che avrà a che fare con questo oggetto e nel farlo imparerò come è fatto h.

# Domanda

**Tra questi modelli, quali sono quelli che rendono sensato cercare di ricostruire h come mezzo per predire y?**

Quando il processo totale è uno tra questi, in alcuni casi è conveniente modellare h per predire y, in altri casi no.

In quali di questi casi è conveniente?



E’ conveniente imparare h per predire y nei modelli 2 e 4.

Modello n.4: h genera y e x. Se conosco h riesco a predire y.

Modello n.2: estraggo y, grazie a y estraggo h e da h genero x. Se imparo h ho qualche informazione su y perché y l’ho usato per generare h.

Modello n.3: y determina x in modo indipendente da h, quindi h non fornisce alcuna informazione su y.

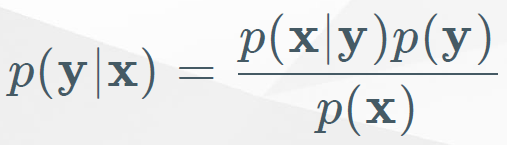
Modello n.1: x determina y però c’è un collegamento tra h e x ma mediato da x. Se imparo bene x senza guardare h ho esattamente tutta l’informazione necessaria per predire y.

Se imparo h posso predire x e quindi anche y ma forse a questo punto lo sforzo aggiuntivo di predire h potrebbe non essere utile. Può essere utile quando h è più semplice da imparare rispetto ad apprendere x.

Analizziamo il modello n.2

Quando y è collegato ad una delle cause di h (caso più semplice quando y è esattamente un attributo di h), allora può essere utile imparare h e da questo determinare subito x.

Per la regola di Bayes:



p(x) è collegata intimamente alla p(y|x) quindi conoscere p(x) può essere molto utile per predire y.

Dopo l’assunzione appena fatta, questo spiega perché apprendere h può essere molto utile per apprendere p(y|x).

|  |
| --- |
| **Nota bene**:  Il caso appena descritto è un modello generativo in quanto si vuole stimare la densità p(y|x) in modo preciso.  Nel caso invece di un modello discriminativo, si vuole predire semplicemente y, allora p(x) si può ignorare perché è costante su tutte le y.  quindi la probabilità sarà molto bassa o molto alta per tutte le y. La p(x) è ignorata. |

# Cosa codificare?

Una difficoltà è decidere cosa codificare in h. E’ spesso impossibile codificare tutti i possibili fattori di variazioni che influenzano un osservazione.

Esistono due principali strategie:

* **Usare un segnale supervisionato** **per guidare il processo non supervisionato**. Nel costruire gli auto-encoder si può aggiungere un classificatore per avere un feedback dagli esempi supervisionati.
* **Usare un h molto grande quando si usa solo l’apprendimento non supervisionato**.

## Concetto di salienza: Problema MSE

Un’altra strategia è **cambiare la definizione di cos’è saliente**, ovvero cosa è importante per la rappresentazione.

Storicamente gli oggetti (es. auto-encoders) sono in genere addestrati usando come misura di loss il **Mean Squared Error (MSE). Potrebbe essere problematico**.

Quando si usa MSE nel caso delle immagini, implicitamente specifica che le cause rilevanti siano soltanto quelle che hanno effetto su un grande numero di pixels.

Una ricerca ha dimostrato che se si addestra un auto-encoder ad apprendere un task un braccio robotico che gioca a ping pong utilizzando MSE, l’auto-encoder fallisce nel modellare una delle informazioni più importanti in assoluto: la pallina da ping pong. Quindi è completamente inutile allo scopo per cui viene addestrato. Ciò succede perché la pallina di ping pong occupa un numero ridotto di pixels sullo schermo e quindi il MSE tende a ignorarla.

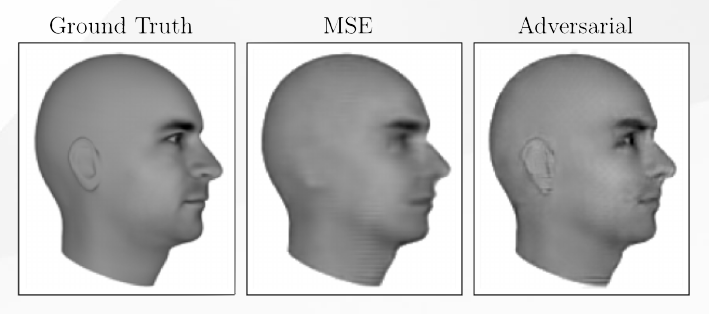
## Introduzione alle GANs

Con l’idea di cambiare la definizione di salienza, si può cambiare anche il modo con cui si valutano gli errori.

Se il gruppo di pixel che occupano poco spazio sullo schermo ma a patto che sia molto riconoscibile, allora si può decidere che quel gruppo di pixel sia più importante. Uno dei modi per implementare questo tipo di strategia è attraverso uno strumento chiamato **GANs (Generative Adversarial Networks)**. La salienza è definita implicitamente da un “gioco” che viene effettuato all’interno delle GANs tra encoder e discriminatore.

Le GANs sono reti che implementano la seguente idea: un encoder prende l’input e cerca di ricostruirlo nel modo più fedele più possibile. Dall’altra parte c’è un discriminatore che, osservando le istanze di input veri e istanze generati dall’encoder, cerca di capire quale delle istanze è vera e quale è stata generata in modo artificiale.  
Esiste quindi un “gioco” tra le due componenti (encoder e discriminator) tale per cui l’encoder cerca di costruire input che inganni il discriminatore e il discriminatore cerca di imparare al meglio possibile come distinguere gli input falsi da quelli veri. Questo tipo di gioco permette di descrivere in modo implicito tante proprietà che sarebbero difficili da descrivere esplicitamente. Quindi questo tipo di gioco riesce a modellare molto meglio delle informazioni sull’importanza (salienza) delle cause che generano i dati che difficilmente si riescono a incorporare in modelli che sono addestrati in modo tradizionale.

# MSE e GANs a confronto



Ground Truth: immagine sottoposta ai sistemi, l’immagine di un volto umano generata automaticamente

**MSE**: Autoencoder addestrato con MSE per ricostruire l’immagine → orecchie cancellate e occhi sfocati. Succede perché l’informazione sull’orecchio umano (cambia di molto la forma e la posizione dell’orecchio nei diversi volti umani). Esiste una multi-modalità dove ciascuna “vetta” rappresenta un orecchio specifico. Il MSE cerca di imparare una rappresentazione che sia comune a tutte e per farlo è costretto a fare una media delle varie orecchie. Siccome i diversi tipi di orecchie sono molto diversi tra di loro, la media fa scomparire l’orecchio in toto. Per l’occhio stesso discorso, la varietà è minore ma si ottiene un occhio sfocato.

**Adversarial**: con le GANs questo non succede perché anziché imparare la media, imparano a generare una delle mode. Capiscono che non è importante modellare tutti gli orecchi specifici, ma modella questo particolare orecchio in questa immagine. Si ottengono risultati molto più soddisfacenti.

# Fattori causali sono robusti

Una delle ragioni per apprendere un modello causale è che **i fattori causali sono robusti ai cambiamenti del mondo**. Spesso si è di fronte a cambiamenti nelle distribuzioni dei dati che osserviamo che possono dipendere da molti fattori: concetto sottostante non è costante dal punto di vista temporale, cambia nel tempo oppure cambia la natura del task che si sta cercando di modellare. In questi casi spesso i fattori causali rimangono gli stessi perché nel mondo fisico tendono a rimanere costanti nel tempo. Cambia la distribuzione che sottende questi fattori causali (quanto è frequente la loro insorgenza) però se si è appreso bene a riconoscere il fattore causale, questo aiuta ad adattarsi velocemente ai cambiamenti.

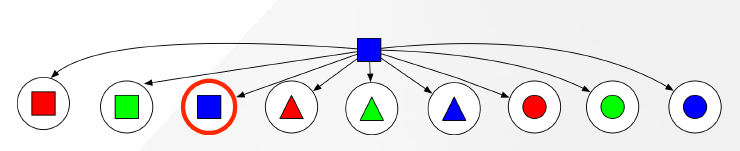
Rappresentazione distribuita

Un aspetto interessante delle reti neurali è che spesso apprendono rappresentazioni distribuite dei dati.

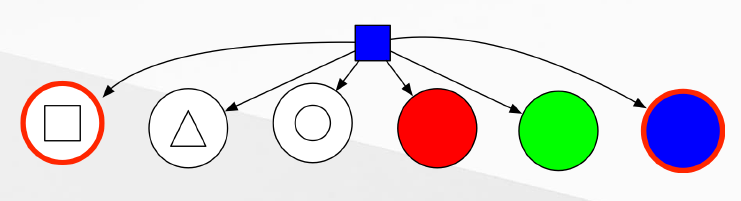
Una **rappresentazione distribuita** è una in cui ogni oggetto è descritto con tante features e ognuna di queste è coinvolta nella rappresentazione di molti oggetti.

La rappresentazione distribuita contiene tante features che vengono condivise tra varie rappresentazioni.

Si consideri il caso in cui un sistema deve apprendere il colore e la forma di qualche oggetto, nell’esempio un quadrato blu. Si possono rappresentare tutte le possibili forme con tutti i possibili colori e poi rappresentare il quadrato blu individuandolo tra tutte le possibili rappresentazioni. Questo equivale ad un one-hot representation: (0 0 1 0 0 0 0 0 0).



Un altro modo per risolvere il problema: avere una rappresentazione delle forme e una rappresentazione separata dei colori.



Il quadrato blu è una rappresentazione che coinvolge un quadrato e un colore blu, ovvero la rappresentazione:

(1 0 0 0 0 1).

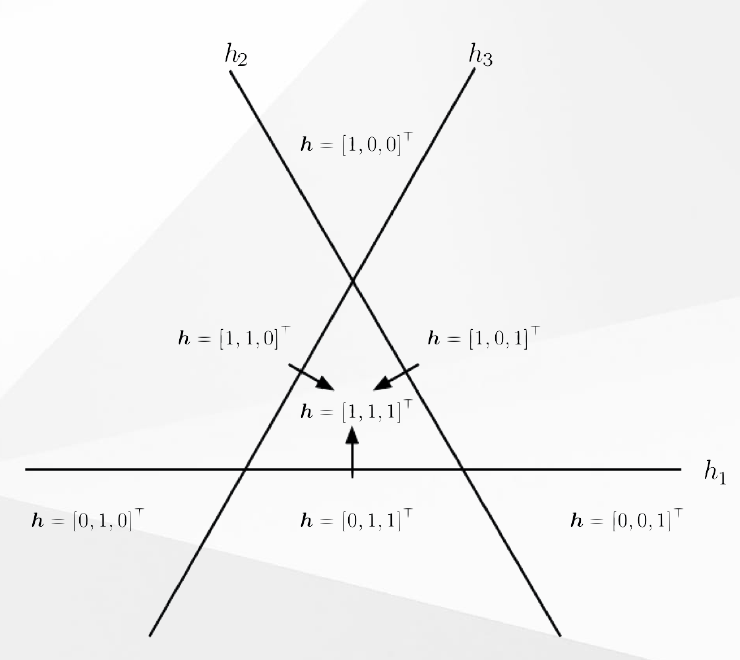
In questo ultimo caso l’oggetto “quadrato blu” è descritto da tante features e ognuna di esse è coinvolta nella descrizione di più oggetti. Il colore blu permette di descrivere un quadrato blu, un triangolo blu, un cerchio blu. Quindi l’idea di una rappresentazione distribuita è avere una rappresentazione che sia una composizione di attributi che si possono comporre tra loro arricchendo di significato il modello.

# Generalizzazione

La rappresentazione distribuita è molto importante in quanto induce spesso una **ricca similarità nello spazio** in cui oggetti **simili dal punto di vista semantico** **sono molto vicini** in termini di distanza euclidea o distanza del coseno dei vettori che li rappresentano. Questa è una caratteristica che manca nelle rappresentazioni puramente simboliche oppure equivalenti alle one-hot representation.

Ad esempio i termini “gatto” e “cane” rappresentati come simboli sono equamente distanti a qualsiasi altra coppia di simboli. Gatto/Casa, Cane/Casa e Gatto/Cane hanno distanza equivalente.

Se invece i tre concetti vengono rappresentati mediante una rappresentazione distribuita (pelliccia, 4 zampe…) allora i due oggetti sarebbero concetti collegati tra loro, hanno una relazione semantica forte fra loro.



**MOLTO IMPORTANTE!**

**Grazie al fatto che si combinano tra di loro,** **le rappresentazioni distribuite riescono a rappresentare un numero esponenziale di concetti**. Si tratta di un vettore di n elementi, ogni elemento può essere 0 oppure 1, con n elementi si possono rappresentare 2^n concetti.

Si supponga di avere tre direttive h1, h2, h3 e le diverse combinazioni di vettori corrispondono a intersezioni degli spazi che descrivono le direttive.

Per h1, la prima componente del vettore assume valore

* 1 → al di sopra di h1
* 0 → al di sotto di h1

Per h2, la seconda componente del vettore assume valore

* 1 → a destra di h2
* 0 → a sinistra di h2

Per h3, la terza componente del vettore assume valore

* 1 → a destra di h3
* 0 → a sinistra di h3

Senza questo tipo di proprietà, per poter apprendere su una rappresentazione senza questa caratteristica e con un numero esponenziale di zone, bisogna assegnare tanti esempi in ciascuna zona e ciò fa crescere in modo esponenziale il numero di esempi utili per l’apprendimento.

Nel momento in cui si trova una rappresentazione che sbroglia le cause delle variazioni, allora queste regioni possono essere apprese con un **numero limitato di attributi** (vedi figura di h1, h2, h3) e quindi si può usare un numero minore di esempi perché per dare supporto statistico ad una certa zona contribuiscono solo gli esempi che condividono parte della rappresentazione. Uno stesso esempio partecipa a più zone e quindi con meno zone si riesce a dare supporto statistico ad ogni zona. Ciò permette di **generalizzare** meglio, ho bisogno di meno dati per dare supporto alle ipotesi.

Un altro argomento circa il motivo per cui **una distribuzione distribuita permette di generalizzare bene** è basato sull’osservazione che, sebbene questi modelli siano in grado di rappresentare molto bene zone molto differenti, la loro **capacità di rappresentazione rimane** comunque **limitata**. Permettono tanta flessibilità, ma non flessibilità infinita. Se dessero flessibilità infinita, si andrebbe verso una situazione in cui è facile essere in presenza di overfitting. Per chiarire meglio l’idea della capacità limitata, bisogna introdurre il concetto di VC Dimension.

# Vapnik-Chervonenkis (VC) Dimension

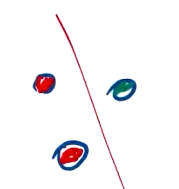
VC Dimension è stato introdotto nell’ambito dell’apprendimento PAC (Probably Approximately Correct), uno degli ambiti formali di studio dell’apprendimento automatico, alla base di tanti risultati che permettono di predire la capacità di generalizzazione di un modello sulla base della VC Dimension.

**La VC Dimension è una misura della capacità di un modello di discriminare gli esempi forniti**.

Si consideri un modello con capacità infinita. Dato in input al modello un qualsiasi dataset, si vuole apprendere un modello in grado di discriminare gli esempi con una qualsiasi etichettatura. Il modello appreso farà overfitting perché è in grado di rappresentare benissimo qualsiasi training set e probabilmente non sarà in grado di estendere questa sua capacità anche al test set.

**La VC dimension** è un modo formale per catturare l’idea che il modello è capace di rappresentare tante possibili funzioni ed **è definita come la** **dimensione del più grande insieme che può essere “frantumato” (shattered) dal modello**.

Un **insieme** si dice **“frantumato” (shattered)** da una famiglia di funzioni F se per ognuna delle possibili etichettature di punti nell’insieme, esiste una funzione in F che può etichettare correttamente tutti i punti.



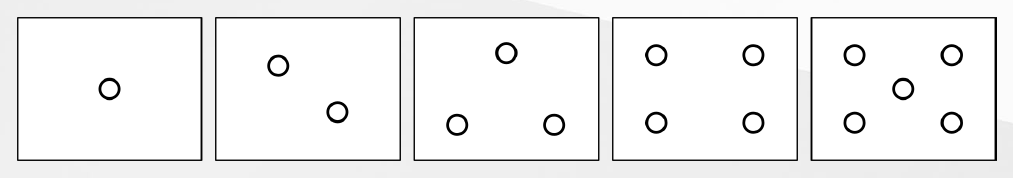
Indipendentemente da come sono etichettati i punti, in F c’è un modo per separarli. La famiglia di funzioni F frantuma questo insieme.

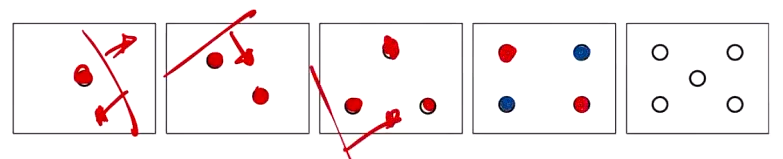
# Domanda

Si consideri l’insieme di tutti i classificatori lineari in .



Quali di questi insiemi sono frantumati da F?





Nota bene: più è alto il numero di punti, più è difficile trovare una funzione in F che riesca a separare i punti per qualsiasi etichettatura.

1, 2, 3 possono essere frantumati

4 e 5 non possono essere frantumati

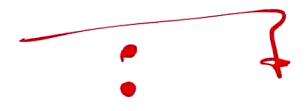
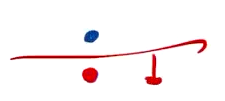
**La VC dimension è definita come la dimensione del più grande insieme che può essere frantumato dal modello**.

# Domanda

Qual è la VC Dimension dell’insieme delle funzioni costanti (tutti i classificatori lineari paralleli all’asse delle x)?

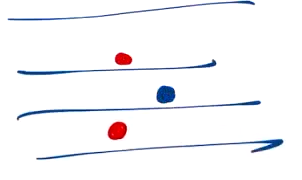
Risposta: 2

Qualsiasi sia l’etichettatura di due punti, si riesce a trovare una soluzione.



Non può essere più di 3 in quanto le funzioni lineari non riescono a frantumare l’insieme di 4. Se non riescono tutte le funzioni lineari a frantumare un insieme composto da più di 3 punti, a maggior ragione neanche le funzioni lineari parallele all’asse x non riescono in quanto sono sottoinsiemi delle funzioni lineari.

Per quanto riguarda 3 punti, non esiste una funzione costante lineare parallela all’asse x che permetta di etichettare tutti e tre i punti correttamente.



# Capacità dei classificatori basati sulle rappresentazioni distribuite

**La VC dimension di una rete neurale con unità di soglia lineari è solo O(w log w)**, dove w è il numero di pesi nella rete.

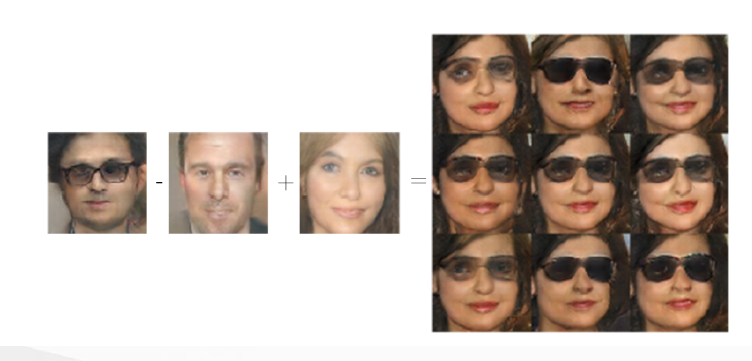
Questo implica che sebbene la rappresentazione sia in grado di distinguere tra molte zone diverse, non tutte le possibili zone possono essere rappresentate. Questo limita la capacità della rete e limita anche i problemi di overfitting, aumentando così la generalizzazione.

# Le rappresentazioni distribuite possono essere interpretabili

Molti autori hanno notato che, quando si apprende una rappresentazione distribuita, spesso si ottiene una rappresentazione le cui features possono essere interpretate, corrispondente ad una etichetta che gli umani potrebbero naturalmente assegnare. Ma questo non è sempre vero.

Infatti l’interpretazione delle rappresentazioni apprese da una rete neurale profonda è un campo di ricerca non ancora risolto.

Esempio: un sistema che apprende una rappresentazione latente tale per cui si possono combinare le rappresentazioni per ottenere risultati semanticamente ragionevoli per un essere umano. Si prende la rappresentazione di un uomo con gli occhiali, si sottrae la rappresentazione di un uomo senza occhiali, dopodiché si aggiunge la rappresentazione di una donna e si ottiene una rappresentazione che si trova in una zona in cui nel suo intorno ci sono questo tipo di immagini (immagini di donne con gli occhiali).



Si riesce ad ottenere quando la rappresentazione distribuita ha anche un senso semantico.

E’ stato quindi dimostrato che un modello generativo può apprendere una rappresentazione di immagini di volti, con direzioni separate nello spazio di rappresentazione catturando diversi fattori di variazione.

# Guadagno esponenziale dalla profondità

È stato dimostrato in molti contesti diversi che organizzare il calcolo attraverso la composizione di molte non linearità e una gerarchia di caratteristiche riutilizzate può dare un impulso esponenziale all'efficienza statistica, oltre all'incremento esponenziale dato dall'utilizzo di una rappresentazione distribuita.

I risultati teorici relativi al potere espressivo delle architetture profonde affermano che esistono famiglie di funzioni che possono essere rappresentate in modo efficiente da un'architettura di profondità k, ma richiederebbero un numero esponenziale di unità nascoste (rispetto alla dimensione di ingresso) con profondità insufficiente (profondità 2 oppure k-1).

Il **teorema di rappresentazione** afferma che **basta un solo livello hidden per rappresentare qualsiasi funzione**. Una rete shallow può rappresentare ciò che vogliamo a patto che le diamo un numero sufficiente di unità a disposizione. Esistono famiglie di funzioni che possono essere rappresentate in modo efficiente con una rete che ha una certa profondità k, ma per le stesse funzioni si volesse rappresentare una rete di lunghezza minore, vi è la necessità di un incremento esponenziale del numero di unità nascoste.

Rete con profondità 5 → 100 neuroni

Rete con profondità 2 → 100.000 neuroni

Si ha un vantaggio esponenziale nell’aumentare la profondità della rete, e questo ha un impatto anche sulla rappresentazione. **Si riesce ad apprendere un ottima rappresentazione con una rete molto profonda, ma per apprendere una rappresentazione ugualmente buona ma con una rete più piatta ho bisogno di molti più esempi e molti più neuroni nella rete.**